

09 / 5 82939

530 Rec'd PCT/PTO 30 6 JUL 2000

VERIFICATION OF TRANSLATION

Re: International Patent Application No. PCT/FR99/02696

I, Carolyn Hopwood (B.A., M.I.L., M.I.T.I.)
154, Chobham Road
London
E15 1LZ

hereby declare that I am the translator of the documents attached
and certify that the following is a true translation to the best of
my knowledge and belief.

Signature of translator ... *C. Hopwood*

Dated this *23rd* day of *June* 20 *00*...

A DATA COMPACTION METHOD FOR AN INTERMEDIATE OBJECT CODE
PROGRAMME EXECUTABLE IN AN ON-BOARD SYSTEM PROVIDED WITH
DATA PROCESSING RESOURCES AND CORRESPONDING ONBOARD SYSTEM
WITH MULTIPLE APPLICATIONS

5

The present invention relates to a method of compacting a programme of the intermediate object code type, which can be run in an on-board system provided with data processing resources, and to the corresponding compaction method.

10 These days, on-board systems provided with data
processing resources enable increasingly complex and larger
numbers of functions to be performed because the hardware
used for these portable objects and their software, or more
specifically the application programmes embedded in the
15 latter as a means of enabling them to run one or more
specific functions, are constantly being optimised. The
concept of on-board system covers any portable computer
system, such as a portable object, microprocessor card or
similar, as opposed to a conventional microcomputer.

20 This is particularly so in the case of microprocessor
cards, also known as chip cards, such as that illustrated in
figure 1a, used in conjunction with a compiler to generate
instructions and an interpreter which enables these
instructions to be run by the microprocessor, as illustrated
25 in figure 1b. Conventionally, as illustrated in figure 1a,
a microprocessor card 10 comprises an input/output system 12
linked to the microprocessor 14, a RAM memory 16, a non-
volatile memory 18 comprising a read only ROM memory 18b and
a programmable memory 18a. All of these elements are linked
30 to the microprocessor 14 by a BUS line. A data
encryption/decryption module 20 may be provided, as
required.

Figure 1c illustrates all the application software

elements, such as electronic purse, electronic commerce or health, embedded in the non-volatile programmable memory, the interpreter in a non-volatile programmable memory or a read only memory and the operating system in a read only memory ROM.

The intermediate object code is generated by the compiler from a source programme, usually written in high level language based on ASCII characters. The source programme and the corresponding intermediate object code may be run by all the standard microprocessors because the interpreter ensures that the standard instructions of the intermediate object code are translated, in software terms, into instructions that can be directly executed by the microprocessor.

By way of example, although this is not restrictive, microprocessor card manufacturers have recently developed interpreters embedded in the read only memory ROM. This type of interpreter reads in sequence a programme or intermediate object code, supporting an application for example, which is loaded into the programmable memory of the microprocessor card, for example. Each standard instruction of this intermediate object code is interpreted by the interpreter, then run by the microprocessor. As a general rule, the standard instructions of the intermediate object code enable changing functions to be processed, such as arithmetic processing and object manipulation. The object concept relates to computed objects such as lists, data tables or similar.

However, due in particular to the fact that these microprocessor cards are portable in nature, the size and space occupied by the latter are limited. The same applies to the size of their programmable memory, which is limited, by construction, to several kilobytes. This structural

002201" 66528560

remarkable because the intermediate object code programme is searched in order to find identical sequences of successive standard instructions and the identical sequences of successive standard instructions are subjected to a comparison test to find a function, based on at least the number of occurrences of these sequences in the intermediate object code programme, that is higher than a reference value. If the above-mentioned test returns a positive response, a specific instruction is generated for each identical sequence of successive standard instructions which satisfies the test step, by defining a specific operating code and associating this specific operating code with the sequence of successive standard instructions. In addition, each occurrence of each sequence of standard successive instructions in the stored intermediate object code programme is replaced by the specific operating code associated with it to obtain a compacted intermediate object code programme, a series of standard instructions and specific operating codes. A decompression table is stored in the memory which enables a reciprocal link to be made between each specific operating code inserted and the sequence of successive standard instructions associated with the latter. This process enables the memory space occupied by the compacted intermediate object programme to be optimised by storing only one occurrence of identical sequences of successive standard instructions in the programmable memory.

The method, the system of compacting an intermediate object code programme and the corresponding multi-application on-board system proposed by this invention may be used in the technical field of on-board systems, more specifically for running and managing microprocessor cards.

They will be more readily understood from the

description below and the appended drawings, in which, apart from figures 1a to 1c which illustrate the prior art,

- figure 2a is a general flow chart illustrating a method of compacting an intermediate object code programme as proposed by this invention;

- figure 2b is a synoptic diagram illustrating how the different operators needed to obtain a compacted intermediate object code programme and parameters enabling this programme to be decompressed or executed, are applied;

- figure 2c, given purely as an illustration, shows this compacted intermediate object code programme embedded in a programmable non-volatile memory of a microprocessor card and the parameters used to decompress and run the latter;

- figure 3a is a diagram of one specific embodiment, although this is not restrictive, of the structure of a first file made up of the parameters for running or decompressing this intermediate object code programme;

- figure 3b is a diagram showing one specific embodiment, although this is not restrictive, illustrating the structure of a second file made up of these parameters for running or decompressing this intermediate object code programme;

- figure 4 shows, as an illustrative example, a compacted intermediate object code programme, as proposed by the invention, embedded in a non-volatile programmable memory of a microprocessor card or on-board multi-application system;

- figure 5 shows an illustrative example of a specific method of compacting an intermediate object code programme in which specific codes, relating to separate applications or intermediate object code programmes, are updated;

- figures 6a and 6b illustrate, in the form of

00582939 102700

functional elements, a system for compacting an intermediate object code programme as proposed by this invention.

The method of compacting an intermediate object code programme as proposed by this invention will now be described with reference to figure 2a. The terms
 5 intermediate object code programme cover any intermediate programme for the purpose of this patent application.

This method will be described with reference to running an on-board system consisting of a microprocessor card of the type illustrated in figure 1a for example, although this
 10 example is not restrictive, this intermediate object code programme being obtained in a conventional manner, as illustrated in figure 1b, a plurality of applications embedded in the programmable memory of the interpreter and the operating system OS in the ROM memory being illustrated
 15 in figure 1c, although again this is not restrictive.

The intermediate object code programme consists of a series of standard instructions which can be run by the microprocessor through the interpreter.

The method of compacting a programme of this type consists in firstly embedding the latter in the programmable memory 18a, searching the intermediate object code programme at a step 1000 as illustrated in figure 2a for identical
 20 sequences of successive standard instructions, these identical sequences being shown by S_i . By identical sequences is meant a sequence of a given number n of octets likely to appear on a repetitive basis in the above-mentioned intermediate object code programme. Accordingly, the rank i of identical sequences indicates separate
 25 sequences for different values of i . Furthermore, the search step 1000 mentioned above consists in determining the number of occurrences N_i of each said identical sequence S_i . At the
 30 end of the search step 1000, a plurality of identical

09582939-102700

sequences S_i will have been found, each sequence S_i being separate, and a number N_i representing the number of occurrences of each of the sequences S_i in the intermediate object code programme.

5 After said step 1000, the compacting method proposed by the invention consists in subjecting, at a step 1001, the identical sequences of successive standard instructions S_i to a comparison test of a function $f(N_i)$ based on at least the number of occurrences N_i associated with an identical
10 sequence S_i . In figure 2a, the comparison test is written:

$$f(N_i) > \text{Ref.}$$

 If the response to the test 1001 is negative, in which case the function of at least the number of occurrences N_i is not greater than the reference value, the test 1001 is
15 applied to the next identical sequence, of rank $i+1$, the index i being incremented at step 1002.

 The steps 1000, 1001 and 1002 illustrated in figure 2a therefore enable a search to be run in the intermediate object code programme for identical sequences or series of
20 octets or, at the very least, a given significant number of these identical sequences, as will be described farther on in the description.

 If the response to said test 1001 is positive, the compacting method proposed by the invention then consists in
25 generating a specific instruction, denoted by IS_i , by defining a specific operating code, denoted by C_i , and associating with this specific operating code the sequence of successive standard instructions which satisfied the test, this being the sequence of successive standard
30 instructions S_i . In figure 2a, the step by which specific instructions are set up is written:

$$IS_i = C_i:S_i.$$

It should be pointed out that the step of defining a

00552299 "102700
002201" 66628560

5

10

15

25

30

the programmable memory 18a, for example. Said storage may be in the non-volatile memory 18, programmable memory 18a or even in the read only memory 18b, although this is not restrictive.

5 As far as said comparison test 1001 is concerned, it should be pointed out that the function of at least the number of occurrences of each identical sequence S_i may be defined so as to optimise the gain in compression thus achieved. In one embodiment, which is not restrictive, this
10 function may be set up so that a comparison is made between the size of each identical sequence of successive standard instructions and a threshold value, expressed as a number of standard instructions, for example.

Figure 2b provides an illustrative example of an
15 operating mode which allows a compacted intermediate object
code programme to be generated using the method proposed by
the invention.

During an initial stage, the creator of the intermediate object code programme sets up a text file containing the source programme. This programme, put together by the latter on the basis of an evolved language, is generally written in ASCII code so that it can be easily read and can contain comments to facilitate understanding on the one hand and development of the latter on the other. The source programme thus created is inserted in a compiler of the conventional type, known as a standard compiler, the purpose of which is to transform each programme line into executable instructions or, at the very least, into instructions which can be interpreted in order to obtain an intermediate object code programme consisting of a standard sequence of instructions that can be interpreted by the interpreter.

The intermediate object code file thus obtained after

the compilation process is placed in a compression system allowing the compacting method to be run as described above with reference to figure 1. This compression system will be explained later in the description.

5 The compaction process applied and described above will then produce a file of interpretable instructions FCC, i.e. the file constituting the compacted intermediate object code programme and the execution file FEX mentioned earlier in the description.

10 The operating mode of the compression system will be described below with reference to a specific example.

Firstly, the compactor system analyses all the standard instructions I_s and draws up a list of all the series of standard instructions existing in the file forming this
15 latter.

If said file contains 1000 octets, for example, the compactor system launches a search procedure on all the series of at least two octets up to a number Q , for example. Said search may be run for series of two octets, then three
20 octets and so on up to Q octets. In a preferred embodiment, the value of the number Q was 500.

Consequently, for every sequence of instructions S_i , made up of a series of standard instructions I_s , the compactor system determines whether this sequence S_i is
25 already contained in the list. If such is the case, the compactor system adds one unit to the number of occurrences N_i of said sequence S_i .

By the end of said search process, the compactor system will have generated a complex list containing all the
30 instruction sequences S_i examined, each sequence having a number of occurrences N_i in the relevant intermediate object code programme assigned to it.

An illustrative table is given below for an

002201 6E628560

1-7-3-5-7-3-7-3-5-7.

0 The successive instruction sequences S_i which occur in
said intermediate object code programme a number of times
greater than or equal to two, are given in the table below.

4 octets	[7-3-5-7]:2		
3 octets	[7-3-5]:2	[3-5-7]:2	
2 octets	[7-3]:3	[3-5]:2	[5-7]:2

The code for specific instructions C_i is determined chronologically on the basis of the first code corresponding to a standard instruction. In a commonplace intermediate object code, there are currently 106 standard instructions and the codes for these instructions fall between 000 and 105. The first specific instruction code C_i can then have the value 106, the second value 107 and so on. Whenever the identical sequences of instructions S_i are replaced by a new specific instruction code C_i , once such an operation has been completed, the list illustrated in the table above is then re-computed.

5

$$1=106-3-106.$$

10

15

30

Following said operation, the single file or, as applicable, the two above-mentioned files, are transmitted to the target system and processed directly by a loading

5 As a non-restrictive example, the file relating to the compacted intermediate object code programme FCC is stored, without processing, in said programmable memory 18a starting at a given address, denoted by ADR-MEM-PGM.

Figure 2c illustrates the embedding on the one hand of the support file for the compacted intermediate object code programme FCC, the execution file FEX and the file TAB-PRO mentioned above, this latter file having been created by the loading programme in the programmable memory 18a of the microprocessor card.

In this drawing, whilst the table of codes for standard instructions I_s is stored at the level of the interpreter in a table TAB-STD, it is in the programmable memory 18a that the execution file FEX and the file TAB-PRO are stored, enabling the address skips to be linked to the corresponding specific instruction codes C_i , these two tables therefore enabling the compacted intermediate object code programme FCC to be run effectively at the level of the microprocessor of the target unit. An executable unit is therefore provided which can be run by the interpreter under conditions that will be explained below.

Before moving on to explain how a compacted intermediate object code programme FCC is run, a detailed description will be given of the structure of the execution files FEX and the file TAB-PRO and the functional relationship between these latter, with reference to figures 3a and 3b.

Figure 3a provides a detailed illustration of the execution file FEX, which, as explained above, has, in addition to the fields for specific codes C_i and instruction sequences S_i , a field to denote the end of macro-instructions, denoted by FM, indicating the end of said sequence. In a preferred although not restrictive embodiment, each specific code C_i may be inserted at the beginning of the field, on one octet for example, after which each corresponding sequence S_i is inserted in a second field of variable length. The end of macro code FM is of the standard type and corresponds to that used in the conventional languages mentioned earlier in the description.

When an execution file FEX is received whose data structure corresponds to that illustrated in figure 3a, for example, the different fields C_i , S_i and FM are processed separately.

Firstly, the specific code C_i of the corresponding specific instruction IS is written to the file TAB-PRO and the sequence of instructions S_i associated with this specific code constituting said specific instruction is written to a file or memory denoted by MEM-SEQ from an address denoted by ADR-1. This code C_i for the corresponding specific instruction is written to the address TAB-PRO + 3 x (CODE-106). In this relation, the address TAB-PRO is specified as being the address at which the file TAB-PRO is opened whilst the value CODE represents the numerical value of the corresponding code C_i . The operating mode illustrated

00582939 102700

in figure 3b corresponds to an address value TAB-PRO which is arbitrarily set at 0, the first specific code allocated having the value 106 and the following successive specific codes allocated having the values 107 and so on. Figure 3b shows only four specific codes 106, 107, 110 and 120 in order to facilitate understanding, the other memory spaces being filled with arbitrary values.

Under these conditions, Adr-i is the first available address in the memory MEM-SEQ, this address corresponding to the address Adr-1 for the first sequence of instructions $S_i = S_1$. From this first address, which constitutes the opening of the file in the memory MEM-SEQ, the sequences of instructions S_i are therefore written in sequence in the order in which they are loaded. The end of macro code FM is also written at the end of the corresponding series.

After said process of writing to the memory MEM-SEQ and after a step to check that the writing process has proceeded correctly, the loading programme writes to the table TAB-PRO after each specific code C_i the value of the address at which the sequence was written in the memory MEM-SEQ. The loading programme then re-computes a new write address for the next sequence S_i of rank i , incremented or decremented depending on the mode used to run through said sequences of instructions S_i .

An execution process of a compacted intermediate object code programme supported by a file FCC as described above and containing specific instructions will now be described with reference to figure 4.

Such a programme is run by means of the interpreter with the aid of an instruction pointer, denoted by PI. In fact, the instruction pointer PI reads the code of the instruction to be carried out, standard instruction I_s or specific instruction IS , and applies this code to the

002207" 6222222222

interpreter which then triggers the actions corresponding to it.

At the start of running a programme, the instruction pointer PI is loaded with the address from which this
5 programme starts, i.e. the address ADR-MEM-PGM.

The interpreter analyses the value of the code read by the instruction pointer PI. As part of this analysis, the latter determines whether this code value corresponds to a standard type code C_s or, if not, to a specific code C_i .
10 This operation is run from the table TAB-STB stored at the level of the interpreter and by associating the standard instruction codes and hence the standard instructions I_s with the execution addresses in its programme.

If the value of the code read is not in the latter
15 table, the interpreter issues a call to read the table TAB-
PRO in order to check for the existence of the code value
read in the latter table. If the code read is no longer in
this latter table, the interpreter will be unable to run the
instruction read and the programme run will be halted,
20 prompting an error message, not illustrated in the flow
chart of figure 4.

In said figure 4, 2000 denotes the start of the execution operation, 2001 the operation by which the instruction pointer PI is initialised on the first programme instruction and 2002 an operation whereby the instruction indicated by the instruction pointer PI is read. This operation in fact corresponds to the process of reading said code value.

In the same manner, at step 2003 in figure 4, a check
30 is run to determine whether the code value read in the table
belongs or does not belong to the standard codes TAB-STB and
whether this code value read belongs to the table TAB-PRO,
this process in effect forming said test 2003, after which

a distinction is made as to whether the instruction INS read is a standard instruction Is or a specific instruction IS. Fig. 4 does not illustrate the situation in which the code read does not belong as explained above and the other of the two tables which generates an error message ,so as not to overload the drawing.

If a positive response is received to said test 2003 and the code read corresponds to a specific instruction, the instruction pointer PI is computed and stored in the stack so that it can then move on to the next instruction. The interpreter reads from the table TAB-PRO the value of the address for the sequence of instructions S_i associated with the specific code C_i read and initialises the value of the instruction pointer PI with this value. All of these operations are shown under reference 2004 in figure 4. Following said step 2004, the interpreter loops back to the step of reading the code, as illustrated in figure 4, by returning to step 2002.

If a negative response is received to test 2003 and the code read corresponds to an instruction of the standard type Is, the interpreter will check in a test step 2005 whether the value for this code corresponds to an end of macro value representing an end of sequence. If such is the case, the value previously stored in the stack memory is extracted and the stack is updated, this value being loaded into the instruction pointer PI. The operation of extracting the value previously stored in the stack constituting a return address, followed by an update of the stack, is illustrated at 2006, the return address being denoted by ADR-RET. After said step 2006, the interpreter loops back to the process at the step where the code value is read, i.e. step 2002. If a negative response is received to test 2005 and the value of the code read corresponds to an instruction of the standard

002207"66666666

5
10
15

20

25

30

procedures. Using a separate stack memory for the specific instructions IS reduces the total memory consumption as compared with using a single stack.

Furthermore, in order to increase the number of specific instructions IS which may be used instead and in place of the limited number of specific instructions between 106 and 255, in the example described earlier in the description by way of illustration, the specific codes C_i can advantageously be encoded on two octets. Under these circumstances, a specific code value such as the value 255 will then indicate coding on two octets.

Finally, the target system, if it is an on-board multi-application system, comprises several compiled and compacted programmes, i.e. several files FCC of the type described above. These programmes must operate independently. This being the case and there being only one interpreter, it runs all the application programmes loaded by the loading programme. If two application programmes use specific instructions, in the embodiment explained earlier in the description, it is possible that the compactor system will assign the same specific code C_i for two series of different instructions.

In order to remedy such a situation and to enable the interpreter to distinguish between the two codes, the fields for the execution file FEX as described above in relation to figure 3a may be supplemented by a third parameter relating to an identification number for the application in question. This identification number will then also be stored for each assigned specific code listed in the table TAB-PRO. This latter parameter is in fact the reference for the programme loaded at the same time as the file containing the table, allowing each specific instruction code C_i to be associated with instruction sequences S_i replaced by these latter for

002201 6628560

the application in question. When the interpreter is running the programme application, it will then be able to single out the specific instructions relating to this application.

Clearly, the process described above enabling a multi-
 5 application on-board system to be run has the disadvantage of increased memory consumption because an additional field has to be allocated to relate back to the application number in question.

A more advantageous process will now be described with
 10 reference to figure 5.

Figure 5 illustrates an on-board system such as a microprocessor card comprising several applications, shown by A_1 to A_k , the values A_1 to A_k in fact constituting identification numbers for each application. To this end,
 15 when compacting any source programme or application with a given identification number A_1 to A_k for example, in accordance with the method proposed by the invention as explained earlier in the description, the target system, i.e. microprocessor card, transmits the contents of the
 20 memory MEM-SEQ along with the corresponding specific codes C_i to the compactor. In effect, the target system re-computes a file of earlier specific coefficients, denoted by F-C-ANT, from the file or table TAB-PRO and the contents of the memory MEM-SEQ, relating to the applications A_1 to A_{k-1} .
 25 The file F-C-ANT ensures that each specific code C_i is reciprocally matched with the sequence S_i associated with it for all the applications A_1 to A_{k-1} . Under these conditions in a simplified but not restrictive embodiment, the file F-C-ANT may consist of a file of the same format as said file
 30 FEX. In the preferred compacting process illustrated in figure 5, the file F-C-ANT of earlier specific codes is then communicated to the compactor in order to teach the latter.

When compressing a new application with an

002207 6662560

identification number A_k , the compactor will look for all occurrences of instruction sequences S_i already registered in the file F-C-ANT, i.e. in fact in the table TAB-PRO of the target system for the earlier applications A_1 to A_{k-1} .

5 Whenever an occurrence is found, the compactor system replaces the corresponding sequence of instructions S_i with the specific code C_i for the corresponding specific instruction IS. Once this operation has been completed, the compactor system can then analyse the application bearing
10 identification code A_k and, of course, look for other occurrences with a view to creating additional specific instructions which have not yet been stored. The file F-C-ANT can then be updated. The decompression process described with reference to figure 5 may advantageously be run in
15 particular as a means of compressing either programmes being loaded on the on-board system for the first time or programmes loaded on the on-board system in addition to other existing compacted programmes.

In either of the two cases above, the compacting process proposed by the invention consists in storing the execution table relating to at least one compacted intermediate object code programme, the first of these programmes in the first hypothesis and one or more existing compacted programmes in the second hypothesis, and then reading the stored execution table for every additional intermediate programme and compressing every additional programme, taking account of the specific instructions and codes stored in the execution table as described earlier in the description. Clearly, the compacted intermediate object code programme thus created can not be run except on the target system, which provided the compactor system with the corresponding relevant file F-C-ANT previously.

When running the method of compacting an intermediate

object code programme, any on-board system such as a multi-application portable object comprising a microprocessor card, for example, and computing resources such as a microprocessor, a programmable memory, a read only memory and a language interpreter will have, as in figure 2c discussed earlier, at least one set of files stored in the programmable memory 18a for example, in addition to the table TAB-STD of standard codes representing an intermediate object code programme stored at the level of the interpreter.

Accordingly, the corresponding portable object will have at least one compacted intermediate object code programme, i.e. the file FCC illustrated in figure 2c. This file may be made up of an application as described above or
15 may be a function such as a data encryption/decryption function or similar. This compacted intermediate object code file will, of course, consist of a series of specific instruction codes C_i and standard instruction codes corresponding to the instruction codes of said intermediate
20 object code programme. The specific instruction codes C_i correspond to sequences of successive standard instructions S_i as mentioned earlier in the description.

Furthermore, as illustrated in figure 2c, an execution table allows a reciprocal link to be applied between each specific operating code C_i and the sequence of successive standard instructions S_i associated with this latter. All of these files enable the memory space occupied in the memory to be optimised, in particular the programmable memory 18a of the portable object.

30 As also illustrated in figure 2c, the execution table has at least one file of successive sequences corresponding to the specific instructions, this file being denoted by the memory MEM-SEQ, and a table, denoted by TAB-PRO, of specific

instruction codes and addresses at which these specific instructions are embedded in the file of successive sequences.

The compacted intermediate object code programme is
5 then run as illustrated in figure 4.

A system for compressing an intermediate object code programme enabling the compression method described earlier in the description to be run will now be explained with reference to figures 6a and 6b.

10 Generally speaking, the compaction method proposed by
the invention will be described as a combination of modules,
these modules being implemented either by hardware means but
preferably by software means, and the data flows between
these modules explained.

15 Figure 6a illustrates the compaction system proposed by
the invention, which comprises at least one module A for
analysing all the directly executable instructions
constituting the intermediate object code programme, shown
by COD-OBJ-INT. Generally speaking, the computer file
20 supporting the intermediate object code programme is
regarded as a string of octets or character string and the
operating mode of the compacting system proposed by this
invention will be looked at from the point of view of the
corresponding processing applied to the string.

25 Starting with said string of octets, the analysis module A is able to read the object code programme COD-OBJ-INT, single out and establish a list of all the standard instruction sequences S_i contained in said programme. In figure 6a, the standard instruction sequences $S_1, S_{i-1}, S_i, S_{i+1}, \dots S_p$, are therefore written in symbolic form in a list in accordance with the symbolic notation of the lists. Accordingly, the analysis module A may consist of a sliding window corresponding to a number n_i of octets, this sliding

30

intermediate object code programme COD-OBJ-INT and the counting command COM-COMP. The compacting module strictly speaking COMP in effect enables every occurrence of every sequence S_i in said intermediate object code programme, considered as a string of octets, corresponding to a specific instruction IS_i to be replaced by the specific code C_i associated with this sequence of instructions.

As regards the operating mode of the compacting module COMP as such, it should be pointed out that it may have a reading sub-module which uses a sliding window, similar to that of the analysis module, enabling the sequence of standard instructions S_i in said string of octets to be located. In practice, when said sequence of standard instructions S_i is located, as illustrated in figure 6a, the compacting module may have a left-hand partitioning and a right-hand partitioning sub-module for the sequence S_i in question in order to generate a left string, denoted by LS, and a right string, denoted by RS. It may then have a concatenation module which uses the specific code C_i constituting the specific instruction IS_i , so that the corresponding specific code C_i , considered as a string of octets, can be concatenated on the one hand, with the left chain LS for example, after which the unit thus formed is concatenated with the right chain RS, which ensures that the sequence S_i will be replaced by the specific code C_i . The compacting module strictly speaking COMP thus issues a compacted intermediate object code programme denoted by COD-OBJ-INT-COMP in figure 6a. Clearly, the compaction system illustrated in figure 6a enables the compaction process described above to be applied to the set of all the directly executable instruction sequences S_i in question.

In one embodiment which is not restrictive, the allocation module AL may, as illustrated in figure 6b, have

Real-time compression tests on the programmes or applications contained in the microprocessor cards sold by BULL CP8 in France have shown that a compression gain of more than 33% can be obtained, which, when applying the compaction process to three applications on a mobile portable object, essentially represents a gain of one additional application for this type of object.

This compression gain was obtained under substantially normal conditions of usage by the user whilst the slowing-down caused by calling up macro instructions is not substantially more than 10% of the running time if no macro instructions are included, this slowing-down being inherent in successively calling up reading operations at the level of the table TAB-B-PRO and the file MEM-SEQ.